Simulation of Shock Waves by Smoothed Particle Hydrodynamics

Mohsen Nejad-Asghar

School of Physics, Damghan University of Basic Sciences, Damghan, Iran

`nasghar@dubs.ac.ir`

ABSTRACT

Isothermal and adiabatic shocks, which are produced from fast expansion of the gas, is simulated with smoothed particle hydrodynamics (SPH). The results are compared with the analytic solutions. The algorithm of the program is explained and the package, which is written in Fortran, is presented in the appendix of this paper. It is possible to change (to complete) the program for a wide variety of applications ranging from astrophysics to fluid mechanics.

Subject headings: Hydrodynamics, methods: numerical, ISM: evolution

1. Introduction

Gas dynamical processes are believed to play an important role in the evolution of astrophysical systems on all length scales. Smoothed particle hydrodynamics (SPH) is a powerful gridless particle method to solve complex fluid-dynamical problems. SPH has a number of attractive features such as its low numerical diffusion in comparison to grid based methods. An adequate scenario for SPH application is the unbound astrophysical problems, especially on the shock propagation (see, e.g., Liu & Liu 2003). In this way, the basic principles of the SPH is written in this paper and the simulation of isothermal and adiabatic shocks are applied to test the ability of this numerical simulation to produce known analytic solutions.

The program is written in Fortran and is highly portable. This package supports only calculations for isothermal and adiabatic shock waves. It is possible to change (to complete) the program for a wide variety of applications ranging from astrophysics to fluid mechanics. The program is written in modular form, in the hope that it will provide a useful tool. I ask only that:

- If you publish results obtained using some parts of this program, please consider acknowledging the source of the package.

- If you discover any errors in the program or documentation, please promptly communicate the to the author.

2. Formulation of Shock Waves

An extremely important problem is the behavior of gases subjected to compression waves. This happens very often in the cases of astrophysical interests. For example, a small region of gas suddenly heated by the liberation of energy will expand into its surroundings. The surroundings will be pushed and compressed. Conservation of mass, momentum, and energy across a shock front is given by the Rankine-Hugoniot conditions (Dyson & Williams 1997)

$$\rho_1 v_1 = \rho_2 v_2 \tag{1}$$

$$\rho_1 v_1^2 + K\rho_1^\gamma = \rho_2 v_2^2 + K\rho_2^\gamma \tag{2}$$

$$\frac{1}{2}v_1^2 + \frac{\gamma}{\gamma-1}K\rho_1^{\gamma-1} = \frac{1}{2}v_2^2 + \frac{\gamma}{\gamma-1}K\rho_2^{\gamma-1} + Q \tag{3}$$

where the equation of state, $p = K\rho^\gamma$, is used. In adiabatic case, we have $Q = 0$, and for isothermal shocks, we will set $\gamma = 1$.

We would interested to consider the collision of two gas sheets with velocities v_0 in the rest frame of the laboratory. In this reference frame, the post-shock will be at rest and the pre-shock velocity is given by $v_1 = v_0 + v_2$, where v_2 is the shock front velocity. Combining equations (1)-(3), we have

$$v_2 = a_0[-\frac{b}{2} + \sqrt{1 + \frac{b^2}{4} + (\gamma-1)(\frac{M_0^2}{2} - q)}] \tag{4}$$

where $a_0 \equiv \gamma K\rho_1^{\gamma-1}$ is the sound speed, $M_0 \equiv v_0/a_0$ is the Mach number, b and q are defined as

$$b \equiv \frac{3-\gamma}{2}M_0 + \frac{\gamma-1}{M_0}q \quad ; \quad q \equiv \frac{Q}{a_0^2}. \tag{5}$$

Substituting (4) into equation (1), density of the post-shock is given by

$$\rho_2 = \rho_1\{1 + \frac{M_0}{[-\frac{b}{2} + \sqrt{1 + \frac{b^2}{4} + (\gamma-1)(\frac{M_0^2}{2} - q)}]}\}. \tag{6}$$

3. SPH Equations

The smoothed particle hdrodynamics was invented to simulate nonaxisymmetric phenomena in astrophysics (Lucy 1977, Gingold & Monaghan 1977). In this method, fluid is represented by N discrete but extended/smoothed particles (i.e. Lagrangian sample points). The particles are overlapping, so that all the physical quantities involved can be treated as continues functions both in space and time. Overlapping is represented by the kernel function, $W_{ab} \equiv W(\mathbf{r}_a - \mathbf{r}_b, h_{ab})$, where $h_{ab} \equiv (h_a + h_b)/2$ is the mean smoothing length of two particles a and b. The continuity, momentum and energy equation of particle a are (Monaghan 1992)

$$\rho_a = \sum_b m_b W_{ab} \tag{7}$$

$$\frac{d\mathbf{v}_a}{dt} = -\sum_b m_b (\frac{p_a}{\rho_a} + \frac{p_b}{\rho_b} + \Pi_{ab}) \nabla_a W_{ab} \tag{8}$$

$$\frac{du_a}{dt} = \frac{1}{2} \sum_b m_b (\frac{p_a}{\rho_a} + \frac{p_b}{\rho_b} + \Pi_{ab}) \mathbf{v}_{ab} \cdot \nabla_a W_{ab} \tag{9}$$

where $\mathbf{v}_{ab} \equiv \mathbf{v}_a - \mathbf{v}_b$ and

$$\Pi_{ab} = \begin{cases} \frac{\alpha v_{sig} \mu_{ab} + \beta \mu_{ab}^2}{\bar{\rho}_{ab}}, & \text{if } \mathbf{v}_{ab} . \mathbf{r}_{ab} < 0, \\ 0, & \text{otherwise,} \end{cases} \tag{10}$$

is the artificial viscosity between particles a and b, where $\bar{\rho}_{ab} = \frac{1}{2}(\rho_a + \rho_b)$ is an average density, $\alpha \sim 1$ and $\beta \sim 2$ are the artificial coefficients, and μ_{ab} is defined as its usual form

$$\mu_{ab} = -\frac{\mathbf{v}_{ab} \cdot \mathbf{r}_{ab}}{\bar{h}_{ab}} \frac{1}{r_{ab}^2/\bar{h}_{ab}^2 + \eta^2} \tag{11}$$

with $\eta \sim 0.1$ and $\bar{h}_{ab} = \frac{1}{2}(h_a + h_b)$. The signal velocity, v_{sig}, is

$$v_{sig} = \frac{1}{2}(c_a + c_b) \tag{12}$$

where c_a and c_b are the sound speed of particles. The SPH equations are integrated using the smallest time-steps

$$\Delta t_a = C_{cour} MIN[\frac{h_a}{\mid \mathbf{v}_a \mid}, (\frac{h_1}{\mid \mathbf{a}_1 \mid})^{0.5}, \frac{u_a}{\mid du_a/dt \mid}, \frac{h_a}{\mid dh_a/dt \mid}, \frac{\rho_a}{\mid d\rho_a/dt \mid}] \tag{13}$$

where $C_{cour} \sim 0.25$ is the Courant number.

4. Results and Prospects

The chosen physical scales for length and time are $[l] = 3.0 \times 10^{16} m$ and $[t] = 3.0 \times 10^{13} s$, respectively, so the velocity unit is approximately $1 km.s^{-1}$. The gravity constant is set $G = 1$ for which the calculated mass unit is $[m] = 4.5 \times 10^{32} kg$. There is considered two equal one dimensional sheets with extension $x = 0.1[l]$, which have initial uniform density and temperature of $\sim 4.5 \times 10^{8} m^{-3}$ and $\sim 10K$, respectively.

Particles with a positive x-coordinate are given an initial negative velocity of Mach 5, and those with a negative x-coordinate are given a Mach 5 velocity in the opposite direction. In adiabatic shock, with $M_0 = 5$, the post-shock density must be 2.9, which is obtained from analytic solution (6) with $Q = 0$ and $\gamma = 2$. The Results of adiabatic shock are shown in Fig. 1-3. In isothermal shock, with $M_0 = 5$, the post-shock density must be 26.9, which is obtained from analytic solution Equ. (6) with $\gamma = 1$. The Results of isothermal shock are shown in Fig. 4-5. Algorithm of the program is shown in Fig. 6.

REFERENCES

Dyson J.E., Williams D.A., 2nd Edition, 1997, *Physics of the Interstellar Medium*, IOP publishing Ltd., p.99

Gingold, R.A., Monaghan, J.J., 1977, *MNRAS*, **181**, 375

Liu, G.R., Liu, M.B., 2003, *Smoothed Particle Hydrodynamics: A Meshfree Particle Method*, World Scientific

Lucy, L.B., 1977, *AJ*, **82**, 1013

Monaghan J.J., 1992, *ARA&A*, **30**, 543

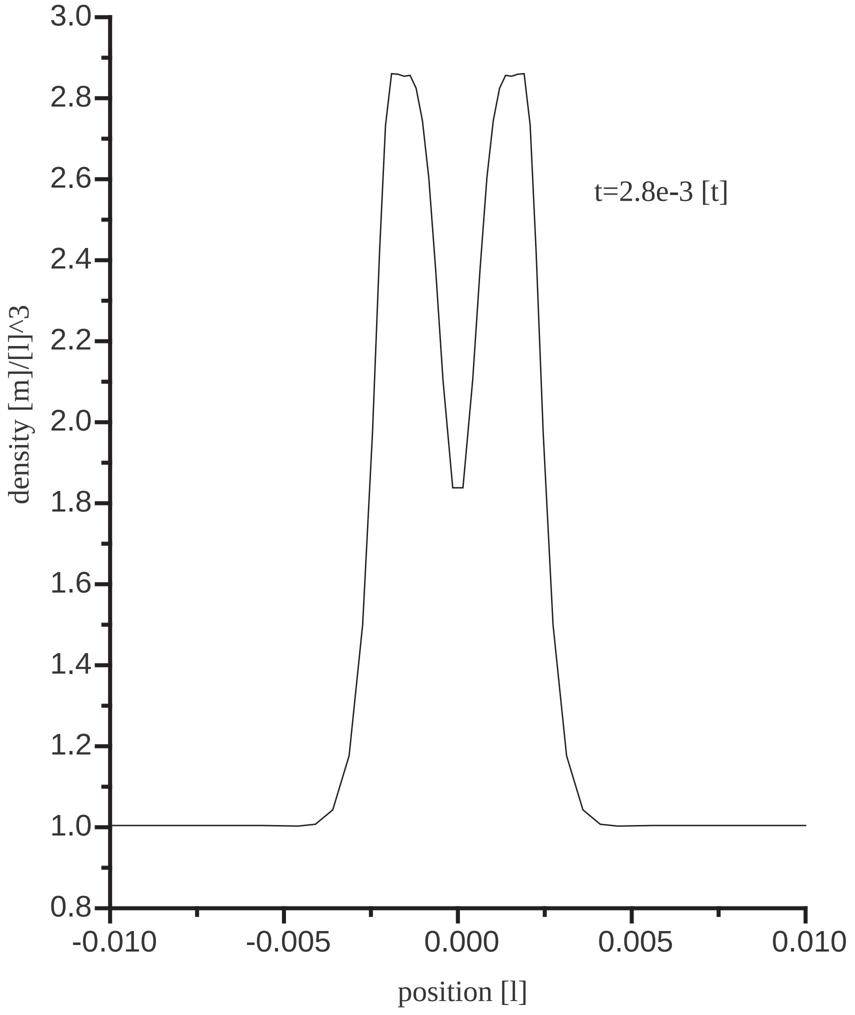

Fig. 1.— The density of adiabatic shock, with $M_0 = 5$, $Q = 0$, and $\gamma = 2$.

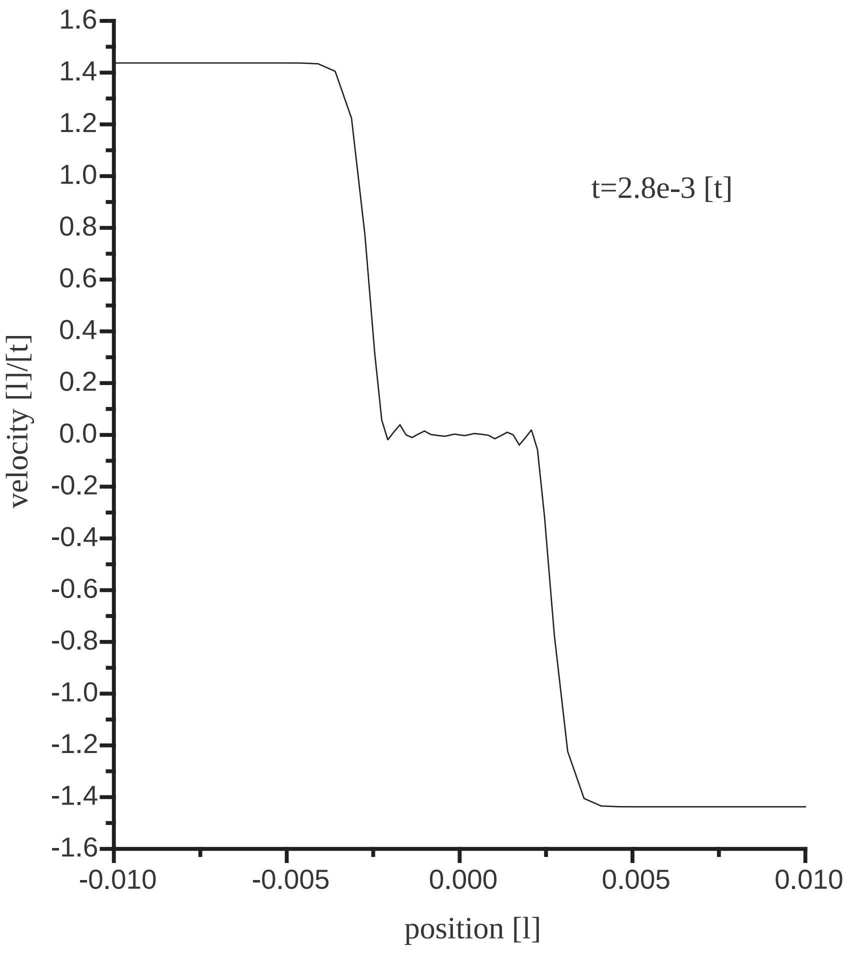

Fig. 2.— The velocity of adiabatic shock, with $M_0 = 5$, $Q = 0$, and $\gamma = 2$.

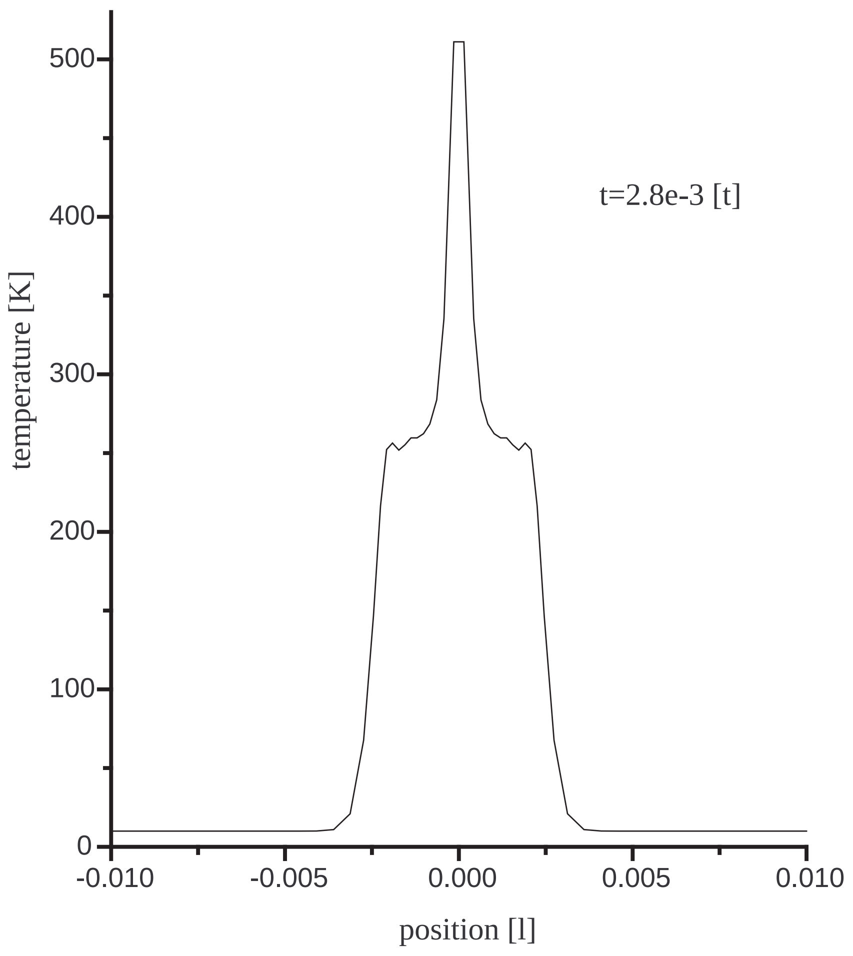

Fig. 3.— The temperature of adiabatic shock, with $M_0 = 5$, $Q = 0$, and $\gamma = 2$.

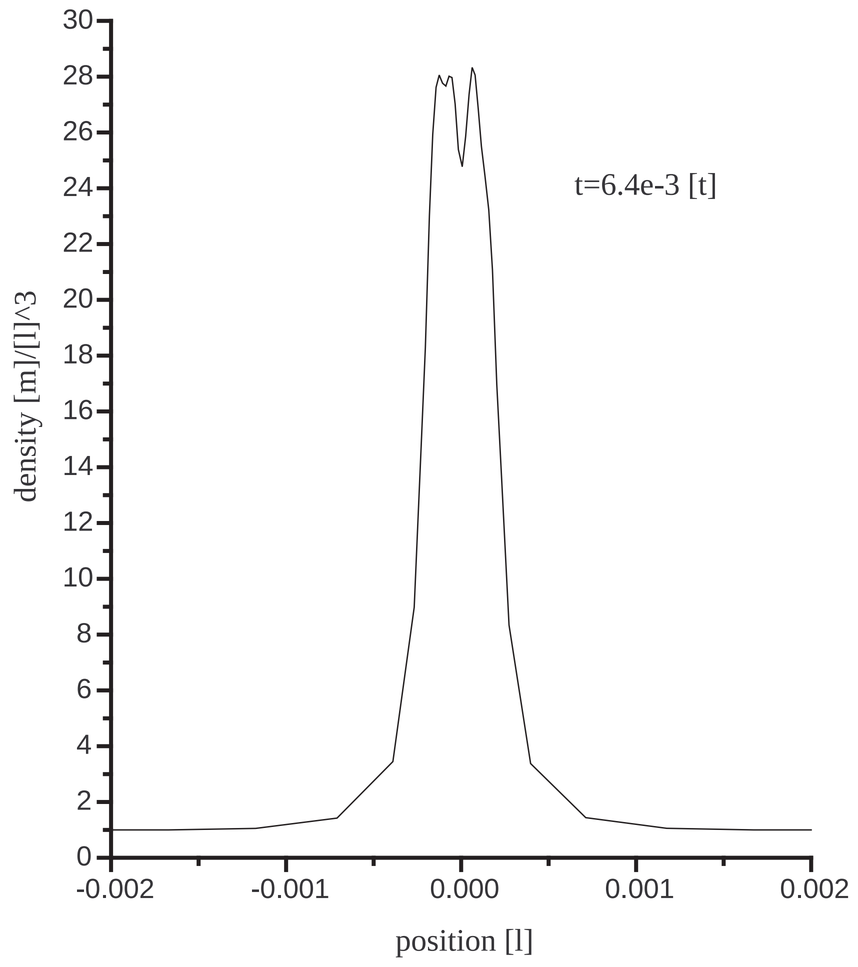

Fig. 4.— The density of isothermal shock, with $M_0 = 5$ and $\gamma = 1$.

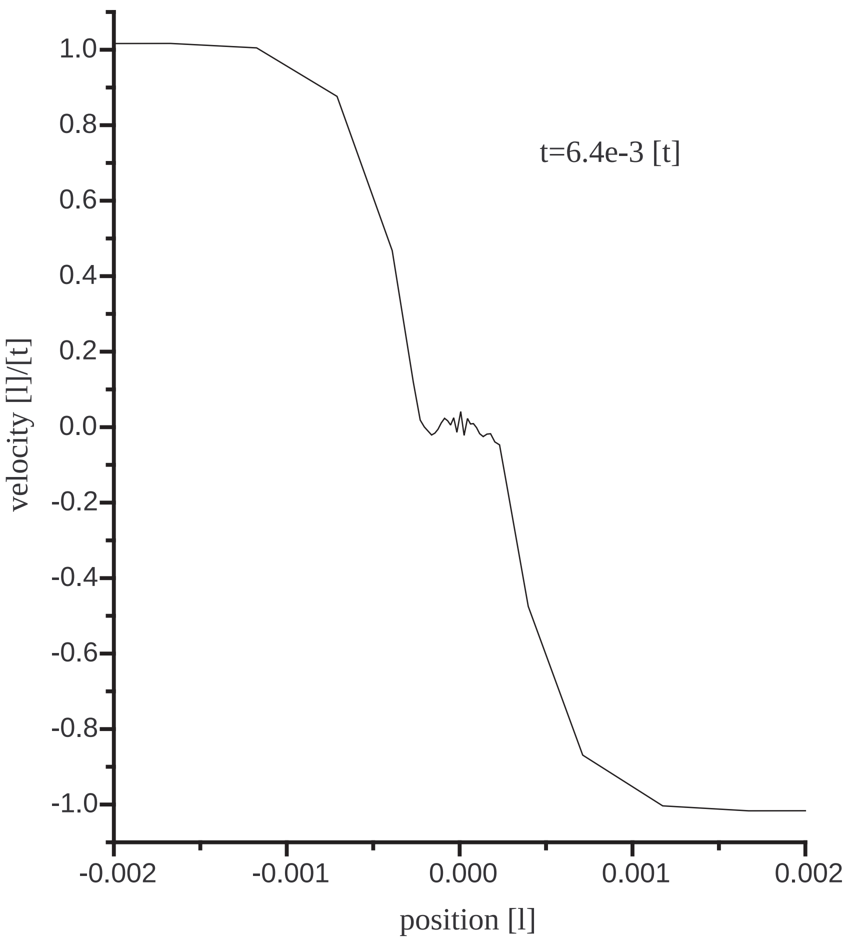

Fig. 5.— The velocity of isothermal shock, with $M_0 = 5$ and $\gamma = 1$.

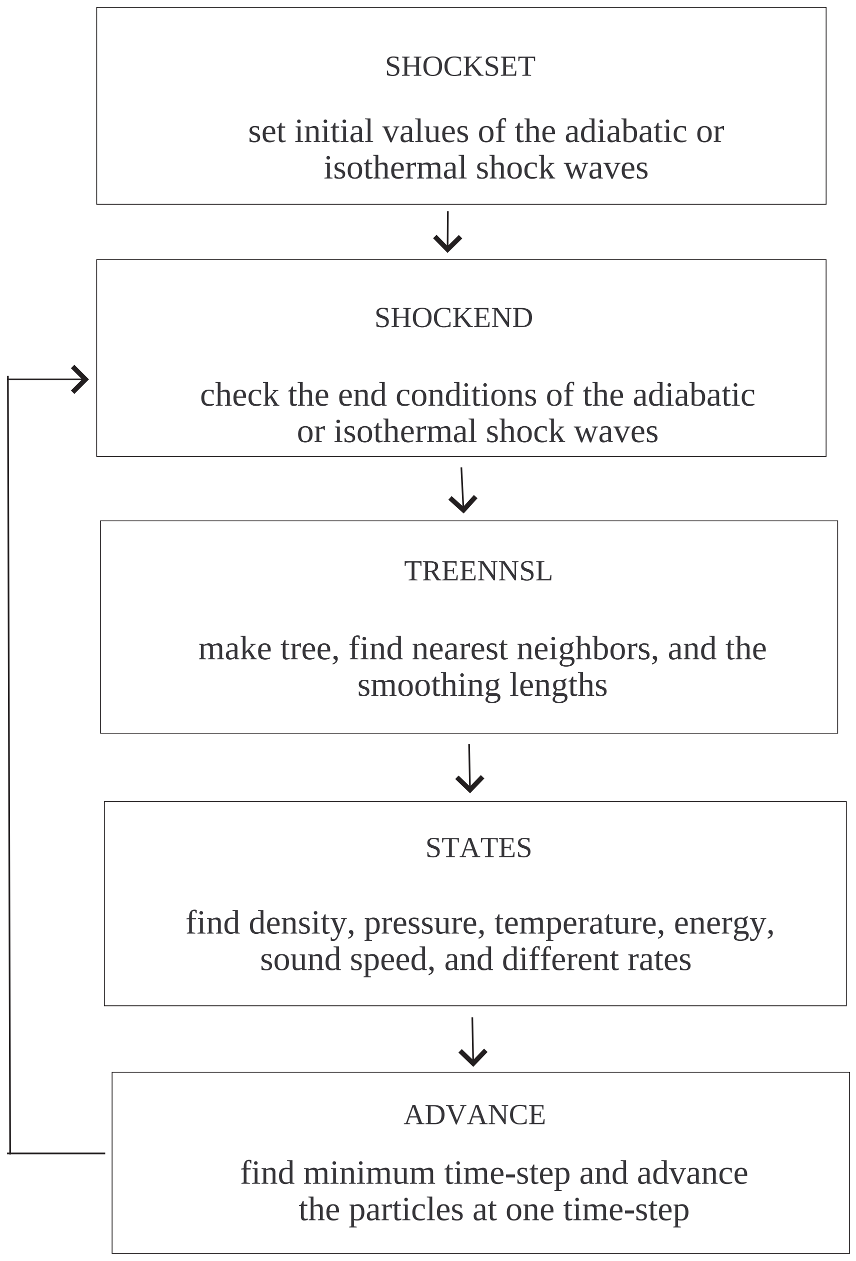
SHOCKSET
set initial values of the adiabatic or isothermal shock waves
SHOCKEND
check the end conditions of the adiabatic or isothermal shock waves
TREENNSL
make tree, find nearest neighbors, and the smoothing lengths
STATES
find density, pressure, temperature, energy, sound speed, and different rates
ADVANCE
find minimum time-step and advance the particles at one time-step

Fig. 6.— Algorithm of the smoothed particle hydrodynamics for simulation of isothermal and adiabatic shocks.

```
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!         This program is provided to simulate the adiabatic
!         and isothermal shock waves
!                        **********************
!                         Mohsen  Nejad-Asghar
!                          nasghar@dubs.ac.ir
!                            January 2006
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      PROGRAM SHOCK
      INCLUDE 'param.inc'
      PRINT*, 'isothermal or adiabatic shock?'
      PRINT*, 'adiabatic=1'
      PRINT*, 'isothermal=2'
      READ(*,*) isorad
      CALL SHOCKSET
      CALL SCENARIOS
      ! investigate the end condition of simulation
 10   CALL SHOCKEND
      ! advance system at one time-step
      CALL ADVANCE
      GOTO 10
      END PROGRAM SHOCK
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE SHOCKSET
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine generates initial particle information for
!     adiabatic or isothermal one dimensional shock
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL extx, delx, mach
        nbody=400
        IF(nbody > maxn)
     &          CALL TERROR('SHOCKSET: SPH number is very large')
        mxcell=nsubc*nbody
        node=nbody+mxcell
        incell=nbody+1
        ! units of length, time, and mass
        ul=3.0e16
        ut=3.0e13
        um=4.5e32
        ! extension of each sheet in x direction (ul)
        extx=0.1
        ! positions of SPH particles
        delx=2.0*extx/nbody
        DO i=1, nbody
          pos(i,1)=extx-i*delx+delx/2.0
        END DO
        ! density of SPH particles (um/ul^3)
        den=1.0
        ! temperature of SPH particles (K)
        temp=10.0
        ! masses of SPH particles (um)
        DO i=1, nbody
          mass(i)=delx*den(i)
        END DO
        ! molecular weight relative to the mass of hydrogen
        xmu=2.0
        ! hydrogen mass
        xmh=1.67e-27
        ! Boltzman constant
        xkb=1.38e-23
        xKK=(xkb/(xmu*xmh))/(ul/ut)**2
        IF(isorad == 1)THEN
          ! polytropic index (adiabatic case)
          gamma=2.0
          ! energy of SPH particles
          u=xKK*temp/(gamma-1)
        ELSEIF(isorad ==2)THEN
          ! polytropic index (isothermal case)
          gamma=1.0
        ENDIF
        ! sound speed
        sound=SQRT(gamma*xKK*temp)
        ! Mach number
        mach=5.0
        IF(isorad == 1)THEN
          ! relative density after simulation for adiabatic case
          cons=SQRT(16.0+((3-gamma)**2+8*(gamma-1))*mach*mach)
          denfinal=(cons+(gamma+1)*mach)/(cons-(3-gamma)*mach)
        ELSEIF(isorad == 2)THEN
          ! relative density after simulation for isothermal case
          cons=SQRT(4.0+mach*mach)
          denfinal=(cons+mach)/(cons-mach)
        ENDIF
        PRINT*,'relative density after simulation must be', denfinal
        ! velocity of SPH particles
        DO i=1, nbody
          IF(pos(i,1) > 0.0)THEN
            vel(i,1)=-sound(i)*mach
          ELSE
            vel(i,1)=+sound(i)*mach
          ENDIF
        END DO
        ! initial smoothing lengths
        diminv=1.0/dim
        DO i=1, nbody
          hh(i)=2.0*(mass(i)/den(i))**diminv
        END DO
        tnow=0.0
        PRINT*, 'setup is successfully completed'
        PAUSE 'press ENTER to continue'
      END SUBROUTINE SHOCKSET
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE SHOCKEND
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine find the end conditions for adiabatic one
!     dimensional shock
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        INTEGER nrev
        nrev=0
        DO i=1, nbody/2
         IF(vel(i,1) > 0.0) nrev=nrev+1
        END DO
        DO i=nbody/2+1, nbody
         IF(vel(i,1) < 0.0) nrev=nrev+1
        END DO
        PRINT*, 'tnow=', tnow,'-----max density=', MAXVAL(den)
        ! stop the program when the reflection waves occur
        IF(nrev > nbody/100)THEN
          CALL SAVEFIG
          PRINT*, 'reversed particles=', nrev
          PAUSE
        ENDIF
      END SUBROUTINE SHOCKEND
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE SAVEFIG
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This program writes particle information into   different
!     files to draw the figures
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        OPEN(1,file='posden.dat')
        OPEN(2,file='pospre.dat')
        OPEN(3,file='posvel.dat')
        OPEN(4,file='posu.dat')
        DO i=1, nbody
          WRITE(1,*) pos(i,1), den(i), tnow
          WRITE(2,*) pos(i,1), pre(i), tnow
          WRITE(3,*) pos(i,1), vel(i,1), tnow
          WRITE(4,*) pos(i,1), temp(i), u(i), tnow
        END DO
        CLOSE(1)
        CLOSE(2)
        CLOSE(3)
        CLOSE(4)
      END SUBROUTINE SAVEFIG
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE SCENARIOS
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine switches for different scenarios in simulation
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
        INCLUDE 'param.inc'
        ! skf--> smoothing kernel function?
        !    =1: Gauss kernel (Gingold & Monaghan 1981)
        !    =2: spline-base kernel (Monaghan 1985)
        !    =3: Quintic kernel (Morris 1997)
        skf=2
        ! nnssl--> nearest neighbors and smoothing length?
        !      =1: fixed smoothing length
        !      =2: variable smoothing length
        nnssl=1
        ! dsm--> density summation method?
        !    =1: summation model without continuity
        !    =2: use continuity equation
        dsm=1
        ! the artificial shear viscosity?
        alphas=1.0
        ! the artificial bulk viscosity?
        betas=2.0
      END SUBROUTINE SCENARIOS
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE ADVANCE
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine advances the particles at one time-step
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL vel0(nbody,dim)
        REAL den0(nbody), hh0(nbody), u0(nbody)
        ! advance particles at first time-step
        IF(tnow == 0.0)THEN
          ! make tree and find neighbors, smoothing
          ! length, and density
          CALL TREENNSL
          ! find all states of the system
          CALL STATE
          ! find minimum time-step
          CALL COURANT
          DO i=1, nbody
            IF(nnssl == 2) hh(i)=hh(i)+hhdot(i)*dtmin/2.0
            IF(dsm == 2) den(i)=den(i)+dendot(i)*dtmin/2.0
            IF(isorad == 1) u(i)=u(i)+udot(i)*dtmin/2.0
            DO j=1, dim
              vel(i,j)=vel(i,j)+acc(i,j)*dtmin/2.0
              pos(i,j)=pos(i,j)+vel(i,j)*dtmin
            END DO
          END DO
          tnow=tnow+dtmin
          RETURN
        ENDIF
        ! advance particles at first half time-step
        DO i=1, nbody
          hh0(i)=hh(i)
          IF(nnssl == 2) hh(i)=hh(i)+hhdot(i)*dtmin/2.0
          den0(i)=den(i)
          IF(dsm == 2) den(i)=den(i)+dendot(i)*dtmin/2.0
          u0(i)=u(i)
          IF(isorad == 1) u(i)=u(i)+udot(i)*dtmin/2.0
          DO j=1, dim
            vel0(i,j)=vel(i,j)
            vel(i,j)=vel(i,j)+acc(i,j)*dtmin/2.0
          END DO
        END DO
        dtmin1=dtmin
        ! make tree and find neighbors, smoothing
        ! length, and density
        CALL TREENNSL
        ! find all states of the system
        CALL STATE
        ! find minimum time-step
        CALL COURANT
        dtmin2=dtmin1/2.0+dtmin/2.0
        ! advance particles at second half time-step
        DO i=1, nbody
          IF(nnssl == 2) hh(i)=hh0(i)+hhdot(i)*dtmin2
          IF(dsm == 2) den(i)=den0(i)+dendot(i)*dtmin2
          IF(isorad == 1) u(i)=u0(i)+udot(i)*dtmin2
          DO j=1, dim
            vel(i,j)=vel0(i,j)+acc(i,j)*dtmin2
            pos(i,j)=pos(i,j)+vel(i,j)*dtmin
          END DO
        END DO
        tnow=tnow+dtmin
        END SUBROUTINE ADVANCE
!====================================================================
!////////////////////////////////////////////////////////////////////
!====================================================================
      SUBROUTINE COURANT
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine evaluates time-step for each particle
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL dt1, dt2, dt3, dt4, dt5, delt(5)
        REAL acc0, vel0
        REAL dt(nbody)
        DO i=1, nbody
          vel0=0.0
          acc0=0.0
          DO k=1, dim
            vel0=vel0+vel(i,k)**2
            acc0=acc0+acc(i,k)**2
          END DO
          vel0=SQRT(vel0)
          acc0=SQRT(acc0)
          dt1=0.0
          dt2=0.0
          dt3=0.0
          dt4=0.0
          dt5=0.0
          IF(vel0 /= 0.0) dt1=hh(i)/vel0
          IF(acc0 /= 0.0) dt2=SQRT(hh(i)/acc0)
          IF(nnssl == 2 .AND. hhdot(i) /= 0.0)
     &                                    dt3=hh(i)/ABS(hhdot(i))
          IF(dsm == 2 .AND. dendot(i) /= 0.0)
     &                                  dt4=den(i)/ABS(dendot(i))
          IF(udot(i) /= 0.0) dt5=u(i)/ABS(udot(i))
          j0=0
          IF(dt1 /= 0.0)THEN
            j0=j0+1
            delt(j0)=dt1
          ENDIF
          IF(dt2 /= 0.0)THEN
            j0=j0+1
            delt(j0)=dt2
          ENDIF
          IF(dt3 /= 0.0)THEN
            j0=j0+1
            delt(j0)=dt3
          ENDIF
          IF(dt4 /= 0.0)THEN
            j0=j0+1
            delt(j0)=dt4
          ENDIF
          IF(dt5 /= 0.0)THEN
            j0=j0+1
            delt(j0)=dt5
          ENDIF
          dt(i)=MAXVAL(delt)
          DO j=1, j0
            dt(i)=MIN(dt(i),delt(j))
          END DO
        END DO
        dtmin=cour*MINVAL(dt)
        IF(dtmin == 0.0) CALL TERROR('COURANT: zero time-step')
      END SUBROUTINE COURANT
!====================================================================
!////////////////////////////////////////////////////////////////////
!====================================================================
      SUBROUTINE TREENNSL
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine makes tree, finds sorted nearest neighbors,
!     and estimates appropriate smoothing length and density of
!     all particles self-consistently.
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        ! construction of tree according to J.E. Barnes
        CALL TREE
        ! find nearest neighbors and smoothing lengths
        CALL NNSL
      END SUBROUTINE TREENNSL
```

```
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE TREE
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine constructs tree, finds its properties, and
!     evaluates the gravitational acceleration
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        ! check particle overlapping
        DO i=1, nbody-1
          DO j=i+1,nbody
            rij=0.0
            DO k=1, dim
              rij=rij+(pos(i,k)-pos(j,k))*(pos(i,k)-pos(j,k))
            END DO
            IF(rij == 0.0) CALL TERROR('TREE: particle overlapping')
          END DO
        END DO
        ! construct the octal-tree body-by-body
        CALL TREELOAD
        ! find the properties of bodies and cells
        CALL TREEPROP
      END SUBROUTINE TREE
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE TREELOAD
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine constructs the octal-tree body-by-body
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        INTEGER MKCELL, p
        ! allocate root cell
        ncell=0
        root=MKCELL()
        ! expand size of the root cell to hold all bodies
        dist=2.05*MAXVAL(ABS(pos))
        clsize(root)=dist
        ! store geometric midpoint of root cell
        DO i =1, dim
          mid(root,i)=0.0
        END DO
        ! load bodies into the new tree, one at a time
        DO p=1, nbody
          CALL LDBODY(p)
        END DO
      END  SUBROUTINE TREELOAD
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE LDBODY(p)
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine loads body p into tree structure
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

        INCLUDE 'param.inc'
        INTEGER p, q, qind, SBINDX, MKCELL, c, p0
        ! set (q,qind) pair in correct subcell of root
        q=root
        qind=SBINDX(p,q)
        ! loop descending tree until an empty subcell is found
 10     IF(subp(q,qind) /= 0)THEN
          ! if subp(q,qind) is a body, extend the tree with a new cell
          IF(subp(q,qind) < incell)THEN
            ! allocate an empty cell to hold both bodies
            c=MKCELL()
            ! locate midpoint of new cell
            DO i=1, dim
              IF(pos(p,i) >= mid(q,i))THEN
                mid(c,i)=mid(q,i)+clsize(q)/4.0
              ELSE
                mid(c,i)=mid(q,i)-clsize(q)/4.0
              ENDIF
            END DO
            ! set size of new cell
            clsize(c)=clsize(q)/2.0
            ! store old body in appropriate subcell within new cell
            p0=subp(q,qind)
            subp(c,SBINDX(p0,c))=p0
            ! link new cell into tree in place of old body
            subp(q,qind)=c
          ENDIF
          ! at this point, the node indexed by (q,qind) is known to
          ! be a cell, so advance to the next level of tree, and loop
          q=subp(q,qind)
          qind=SBINDX(p,q)
          GOTO 10
        ENDIF
        ! found place in tree for p, so store it there
        subp(q,qind)=p
      END SUBROUTINE LDBODY
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      INTEGER FUNCTION MKCELL()
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This function allocates a cell and returns its index
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        ! check remaining space for a new cell
        IF(ncell > mxcell)THEN
          CALL TERROR('MKCELL: no more memory')
        ENDIF
        ! increment cell counter, initialize new cell pointer
        ncell=ncell+1
        MKCELL=ncell+nbody
        ! zero pointers to subcells of new cell
        DO i=1, nsubc
          subp(MKCELL,i)=0
        END DO
      END FUNCTION MKCELL
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      INTEGER FUNCTION SBINDX(p,q)
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This function computes subcell index for node p within cell q
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        INTEGER p, q
        ! initialize subindex to point to lower left subcell
        SBINDX=1
        ! loop over all spatial dimensions
        DO i=1, dim
          IF(pos(p,i) >= mid(q,i)) SBINDX=SBINDX+2**(dim-i)
        END DO
      END FUNCTION SBINDX
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE TREEPROP
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine checks tree structure, assigns critical radius
!     for each cell, computes cell masses, c.m. positions, and
!     quadrupole moments
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        INTEGER p, q
        REAL pos0(dim), dist2
        ! list cells in order of descending size
        CALL SORTLIST
        ! loop processing cells from smallest to root
        DO i=ncell, 1, -1
          p=sortind(i)
          ! check that p is a cell
          IF(p < incell) CALL TERROR('TREEPROP: wrong cell')
          ! zero accumulators for this cell
          mass(p)=0.0
          pos0=0.0
          ! compute cell properties as sum of properties
          ! of its subcells
          DO j=1, nsubc
            q=subp(p,j)
            ! only access cells which exist
            IF (q /= 0)THEN
              ! sum properties of subcells to obtain
              ! values for cell p
              mass(p)=mass(p)+mass(q)
              DO k=1, dim
```

```
                pos0(k)=pos0(k)+mass(q)*pos(q,k)
              END DO
            ENDIF
          END DO
          ! normalize center of mass coordinates by total cell mass
          DO j=1, dim
            pos0(j)=pos0(j)/mass(p)
          END DO
          ! check tree, compute cm-to-mid distance
          ! and assign cell position
          dist2=0.0
          DO j=1, dim
            IF(pos0(j) < mid(p,j)-clsize(p)/2.0 .OR.
     &         pos0(j) >= mid(p,j)+clsize(p)/2.0)
     &            CALL TERROR('TREEPROP: tree structure error')
            dist2=dist2+(pos0(j)-mid(p,j))**2
            ! copy cm position to cell. This overwrites the midpoint
            pos(p,j)=pos0(j)
          END DO
          ! assign critical radius for cell, adding offset
          ! from midpoint for more accurate forces. This
          ! overwrites the cell size
          rcrit2(p)=(clsize(p)/theta+SQRT(dist2))**2
          ! compute quadrupole moments
          DO j=1, nquad
              quad(p,j)=0.0
          END DO
          ! loop over descendants of cell p
          DO j=1, nsubc
            q=subp(p,j)
            IF(q /= 0)THEN
              ! sum properties of subcell q to
              ! obtain values for cell p
              DO m=1, MIN(2,dim)
                DO n=m, dim
                  l=(m-1)*(dim-1)+n
                  quad(p,l)=quad(p,l)+3.0*mass(q)*(pos(q,m)
     &                      -pos(p,m))*(pos(q,n)-pos(p,n))
                  IF(m == n)THEN
                    DO k=1, dim
                      quad(p,l)=quad(p,l)-mass(q)*(pos(q,k)
     &                          -pos(p,k))**2
                    END DO
                  ENDIF
                  ! if q itself is a cell, add its moments too
                  IF(q >= incell) quad(p,l)=quad(p,l)+quad(q,l)
                END DO
              END DO
            ENDIF
          END DO
        END DO
      END  SUBROUTINE TREEPROP
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE SORTLIST
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine sorts cells from largest (root) to smallest
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        INTEGER facell, lacell, nacell
        ! start scan with root as only active cell
        sortind(1)=root
        facell=1
        lacell=1
        ! loop while active cells to process
 10     IF(facell <= lacell)THEN
          ! start counting active cells in next iteration
          nacell=lacell
          ! loop over subcells of each active cell
          DO i=1, nsubc
            DO j=facell, lacell
              ! add all cells on next level to active list
              IF(subp(sortind(j),i) >= incell)THEN
                nacell=nacell+1
                IF(nacell > maxn*nsubc)
     &                  CALL TERROR('SORTLIST: overflow')
                sortind(nacell)=subp(sortind(j),i)
              ENDIF
            END DO
          END DO
          ! advance first and last active cell indices, and loop
          facell=lacell+1
          lacell=nacell
          GOTO 10
        ENDIF
        ! above loop should list all cells; check the count
        IF(nacell /= ncell)THEN
          WRITE(*,*) nacell, ncell
          CALL TERROR('SORTLIST: inconsistent cell count')
        ENDIF
      END SUBROUTINE SORTLIST
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE NNSL
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine finds the nearest neighbors and smoothing
!     length of particles
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL DIST
        INTEGER p, q, qsub
        IF(nnssl == 1)THEN
          ! fixed smoothing length
          DO p=1, nbody
            hh(p)=1.5*(mass(p)/den(p))**(1.0/dim)
            ! effective radius according to hh
            IF(skf == 1) rp=3.0*hh(p)
            IF(skf == 2) rp=2.0*hh(p)
            IF(skf == 3) rp=3.0*hh(p)
            neighb(p)=0
            ! loop processing cells from root to smallest
            DO i=1, ncell
              q=sortind(i)
              rr=rp+clsize(q)
              IF(DIST(p,q,rr,0) < 0.0)THEN
                ! accepted: permit descent
                DO j=1, nsubc
                  qsub=subp(q,j)
                  IF(qsub /= 0 .AND. qsub < incell)THEN
                    ! a body: skip self-consideration
                    IF(qsub /= p)THEN
                      ! test its spacing
                      rr=rp
                      IF(DIST(p,qsub,rr,1) < 0.0)THEN
                        ! accepted as a nearest neighbor
                        neighb(p)=neighb(p)+1
                        ! check number of nearest neighbors
                        IF(neighb(p) == nbody)
     &                        CALL TERROR('NNSL: too many')
                        neighblist(p,neighb(p))=qsub
                      ENDIF
                    ENDIF
                  ENDIF
                END DO
              ENDIF
            END DO
            CALL SORTNEIGHB(p)
          END DO
        ELSEIF(nnssl == 2)THEN
          ! variable smoothing length
          DO p=1, nbody
            numiter=0
 10         numiter=numiter+1
            IF(numiter > 20)THEN
              WRITE(*,*) p, dennew, hhnew
              pause
              CALL TERROR('NNSL: too many iteration')
            ENDIF
            ! first use the smoothing length at this time, which
            ! is advanced via dh/dt=(h/dim)*divvel and find
            ! effective radius according to this hh
            IF(skf == 1) rp=3.0*hh(p)
            IF(skf == 2) rp=2.0*hh(p)
            IF(skf == 3) rp=3.0*hh(p)
            neighb(p)=0
            ! loop processing cells from root to smallest
            DO i=1, ncell
              q=sortind(i)
              rr=rp+clsize(q)
              IF(DIST(p,q,rr,0) < 0.0)THEN
                ! accepted: permit descent
                DO j=1, nsubc
                  qsub=subp(q,j)
                  IF(qsub /= 0 .AND. qsub < incell)THEN
                    ! a body: skip self-consideration
                    IF(qsub /= p)THEN
                      ! test its spacing
                      rr=rp
                      IF(DIST(p,qsub,rr,1) < 0.0)THEN
                        ! accepted as a nearest neighbor
                        neighb(p)=neighb(p)+1
                        ! check number of nearest neighbors
                        IF(neighb(p) == nbody)
     &                        CALL TERROR('NNSL: too many')
                        neighblist(p,neighb(p))=qsub
                      ENDIF
                    ENDIF
                  ENDIF
                END DO
              ENDIF
            END DO
            ! next find density by a summation over the particles
            hmin=hh(p)
            dennew=mass(p)*W(p,p)
            DO jcursor=1, neighb(p)
              j=neighblist(p,jcursor)
              dennew=dennew+mass(j)*W(p,j)
            END DO
            ! change the smoothing length via the
            ! proportionally (1/den)^(1/dim)
            hhnew=2.0*(mass(p)/dennew)**(1.0/dim)
            ! check convergence of smoothing length
            hfrac=ABS(hhnew-hh(p))/hh(p)
            IF(hfrac > 0.01)THEN
              hh(p)=hhnew
              GOTO 10
            ENDIF
            CALL SORTNEIGHB(p)
          END DO
        ENDIF
      END SUBROUTINE NNSL
!====================================================================
!////////////////////////////////////////////////////////////////////
!====================================================================
      FUNCTION DIST(i,q,rr,mode)
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This function estimates the spacing criterion between particle
!     p and node q
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL DIST, rpq
        INTEGER q
        rpq=0
        IF(mode == 1)THEN
          DO j=1, dim
            a=pos(q,j)-pos(i,j)
            rpq=rpq+a*a
          END DO
          DIST=rpq-rr*rr
        ELSE
          DO j=1, dim
            a=ABS(pos(q,j)-pos(i,j))
            rpq=MAX(rpq,a)
          END DO
          DIST=rpq-rr
        ENDIF
      END FUNCTION DIST
!====================================================================
!////////////////////////////////////////////////////////////////////
!====================================================================
      SUBROUTINE SORTNEIGHB(i)
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine sorts the nearest neighbors at ascending
!     distance to particle i
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL rij(neighb(i))
        INTEGER indx(neighb(i)),indxn(neighb(i))
        rij=0.0
        DO jcursor=1, neighb(i)
          j=neighblist(i,jcursor)
          DO k=1, dim
            rij(jcursor)=rij(jcursor)+(pos(i,k)-pos(j,k))**2
          END DO
          rij(jcursor)=SQRT(rij(jcursor))
        END DO
        CALL  INDEXX(neighb(i),rij,indx)
        DO j=1, neighb(i)
          indxn(j)=neighblist(i,indx(j))
        END DO
        DO j=1, neighb(i)
          neighblist(i,j)=indxn(j)
        END DO
      END SUBROUTINE SORTNEIGHB
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE INDEXX(n,arr,indx)
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine indexes an arry arr(1:n), i.e. output the
!     array indx(1:n) such that arr(indx(j)) is in ascending order
!     for j=1,2,..,n. According to 'Numerical Recipes', Press et al.
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        INTEGER n,indx(n),M,NSTACK
        REAL arr(n)
        PARAMETER (M=7,NSTACK=50)
        INTEGER i,indxt,ir,itemp,j,jstack,k,l,istack(NSTACK)
        REAL a
        DO j=1,n
          indx(j)=j
        END DO
        jstack=0
        l=1
        ir=n
1       IF(ir-l.lt.M)THEN
          DO j=l+1,ir
            indxt=indx(j)
            a=arr(indxt)
            DO i=j-1,l,-1
              IF(arr(indx(i)).le.a) GOTO 2
              indx(i+1)=indx(i)
            END DO
            i=l-1
2           indx(i+1)=indxt
          END DO
          IF(jstack.eq.0) RETURN
          ir=istack(jstack)
          l=istack(jstack-1)
          jstack=jstack-2
        ELSE
          k=(l+ir)/2
          itemp=indx(k)
          indx(k)=indx(l+1)
          indx(l+1)=itemp
          IF(arr(indx(l)).gt.arr(indx(ir)))THEN
            itemp=indx(l)
            indx(l)=indx(ir)
            indx(ir)=itemp
          ENDIF
          IF(arr(indx(l+1)).gt.arr(indx(ir)))THEN
            itemp=indx(l+1)
            indx(l+1)=indx(ir)
            indx(ir)=itemp
          ENDIF
          IF(arr(indx(l)).gt.arr(indx(l+1)))THEN
            itemp=indx(l)
            indx(l)=indx(l+1)
            indx(l+1)=itemp
          ENDIF
          i=l+1
          j=ir
          indxt=indx(l+1)
          a=arr(indxt)
3         CONTINUE
          i=i+1
          IF(arr(indx(i)).lt.a) GOTO 3
4         CONTINUE
```

```
          j=j-1
          IF(arr(indx(j)).gt.a) GOTO 4
          IF(j.lt.i) GOTO 5
          itemp=indx(i)
          indx(i)=indx(j)
          indx(j)=itemp
          GOTO 3
5         indx(l+1)=indx(j)
          indx(j)=indxt
          jstack=jstack+2
          IF(jstack.gt.NSTACK) PAUSE 'NSTACK too small in indexx'
          IF(ir-i+1.ge.j-l)THEN
            istack(jstack)=ir
            istack(jstack-1)=i
            ir=j-1
          ELSE
            istack(jstack)=j-1
            istack(jstack-1)=l
            l=i
          ENDIF
        ENDIF
        GOTO 1
      END SUBROUTINE INDEXX
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE STATE
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine finds different states
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        IF(dsm == 1) CALL DENSUM
        CALL DIVVELO
        ! find density rate and smoothing length rate
        IF(dsm == 2) dendot=-den*divvel
        ! find smoothing length rate
        IF(nnssl == 2) hhdot=(hh/dim)*divvel
        ! find pressure, sound speed, and energy of particles
        CALL PRESOUNDENG
        ! find time-rate of velocity (acceleration) and
        ! energy of particles
        CALL RATES
      END SUBROUTINE STATE
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE DENSUM
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine estimates the density via normalization
!     summation method
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL sumW(nbody)
        ! firstly integration of the kernel
        ! secondly density integration
        DO i=1, nbody
          hmin=hh(i)
          den(i)=mass(i)*W(i,i)
          DO jcursor=1, neighb(i)
            j=neighblist(i,jcursor)
            hmin=(hh(i)+hh(j))/2.0
            den(i)=den(i)+mass(j)*W(i,j)
          END DO
        END DO
        DO i=1, nbody
          hmin=hh(i)
          sumW(i)=mass(i)*W(i,i)/den(i)
          DO jcursor=1, neighb(i)
            j=neighblist(i,jcursor)
            hmin=(hh(i)+hh(j))/2.0
            sumW(i)=sumW(i)+mass(j)*W(i,j)/den(j)
          END DO
        END DO
        ! thirdly normalized density
        DO i=1, nbody
          den(i)=den(i)/sumW(i)
        END DO
      END SUBROUTINE DENSUM
!=====================================================================
!/////////////////////////////////////////////////////////////////////
```

```
!=====================================================================
      SUBROUTINE DIVVELO
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine computes velocity divergence for particle i
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL vji(dim), rij(dim), rij0, gradW
        divvel=0.0
        DO i=1, nbody
          DO jcursor=1, neighb(i)
            j=neighblist(i,jcursor)
            hmin=(hh(i)+hh(j))/2.0
            rij0=0.0
            DO k=1, dim
              vji(k)=vel(j,k)-vel(i,k)
              rij(k)=pos(i,k)-pos(j,k)
              rij0=rij0+rij(k)*rij(k)
            END DO
            rij0=SQRT(rij0)
            vdotdelW=0.0
            DO k=1, dim
              gradW=dW(i,j)*rij(k)/rij0
              vdotdelW=vdotdelW+vji(k)*gradW
            END DO
            divvel(i)=divvel(i)+mass(j)*vdotdelW
          END DO
          divvel(i)=divvel(i)/den(i)
        END DO
      END SUBROUTINE DIVVELO
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE PRESOUNDENG
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine estimates the pressure, sound speed, and
!     temperature of particles
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        ! molecular weight relative to the mass of hydrogen
        xmu=2.0
        ! hydrogen mass
        xmh=1.67e-27
        ! Boltzman constant
        xkb=1.38e-23
        xKK=(xkb/(xmu*xmh))/(ul/ut)**2
        IF(isorad == 1)THEN
          ! polytropic index (adiabatic case)
          gamma=2.0
          pre=(gamma-1)*den*u
          temp=(gamma-1)*u/xKK
          sound=SQRT(gamma*xKK*temp)
        ELSEIF(isorad == 2) THEN
          ! polytropic index (isothermal case)
          gamma=1.0
          pre=gamma*xKK*temp*den
        ENDIF
      END SUBROUTINE PRESOUNDENG
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE RATES
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine computes time-rate of velocity (acceleration)
!     and energy of particles
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL rij(dim), gradW, vij(dim), muij
        DO i=1, nbody
          DO k=1, dim
            acc(i,k)=0.0
          END DO
          udot(i)=0.0
          DO jcursor=1, neighb(i)
            j=neighblist(i,jcursor)
            hmin=(hh(i)+hh(j))/2.0
            denmin=(den(i)+den(j))/2.0
            vsig=(sound(i)+sound(j))/2.0
            rij0=0.0
            vijrij=0.0
            DO k=1, dim
              vij(k)=vel(i,k)-vel(j,k)
              rij(k)=pos(i,k)-pos(j,k)
              rij0=rij0+rij(k)**2
              vijrij=vijrij+vij(k)*rij(k)
            END DO
            rij0=SQRT(rij0)
            IF(vijrij < 0.0)THEN
              muij=vijrij*hmin/(rij0**2+(eta*hmin)**2)
              phiij=(-alphas*vsig*muij+betas*muij**2)/denmin
            ELSE
              phiij=0.0
            ENDIF
            preden=pre(i)/den(i)**2+pre(j)/den(j)**2
            rdotdelW=0.0
            vdotdelW=0.0
            DO k=1, dim
              gradW=dW(i,j)*rij(k)/rij0
              acc(i,k)=acc(i,k)-mass(j)*(preden+phiij)*gradW
              rdotdelW=rdotdelW+rij(k)*gradW
              vdotdelW=vdotdelW+vij(k)*gradW
            END DO
            udot(i)=udot(i)+0.5*mass(j)*(preden+phiij)*vdotdelW
          END DO
        END DO
      END SUBROUTINE RATES
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      FUNCTION W(i,j)
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This function evaluates the kernel of particles i and j
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL sig, rij, q, W
        rij=0.0
        DO k=1, dim
          rij=rij+(pos(i,k)-pos(j,k))**2
        END DO
        rij=SQRT(rij)
        htest=(hh(i)+hh(j))/2.0
        IF(hmin /= hh(i) .AND. hmin /= htest)
     &               CALL TERROR('KERNEL: hmin is inconsistent')
        q=rij/hmin
        ! Gauss kernel
        IF(skf == 1)THEN
          sig=(1.0/3.14)**(dim/2.0)
          IF(q <= 3.0)THEN
            W=EXP(-q*q)
          ELSE
            W=0.0
          ENDIF
          W=sig*W/hmin**dim
        ! spline-base kernel
        ELSEIF(skf == 2)THEN
          IF(dim ==1) sig=2.0/3.0
          IF(dim ==2) sig=10.0/(7.0*3.14)
          IF(dim ==3) sig=1.0/3.14
          IF(q <= 1.0)THEN
            W=1.0-1.5*q**2+0.75*q**3
          ELSEIF(q <= 2.0)THEN
            W=0.25*(2.0-q)**3
          ELSEIF(q > 2.0)THEN
            W=0.0
          ENDIF
          W=sig*W/hmin**dim
        ! Quintic kernel
        ELSEIF(skf == 3)THEN
          IF(dim ==1) sig=1.0/120.0
          IF(dim ==2) sig=7.0/(480.0*3.14)
          IF(dim ==3) sig=1.0/(120.0*3.14)
          IF(q <= 1.0)THEN
            W=(3.0-q)**5-6.0*(2.0-q)**5+15.0*(1.0-q)**5
          ELSEIF(q <= 2.0)THEN
            W=(3.0-q)**5-6.0*(2.0-q)**5
          ELSEIF(q <= 3.0)THEN
            W=(3.0-q)**5
          ELSEIF(q > 3.0)THEN
            W=0.0
          ENDIF
          W=sig*W/hmin**dim
```

```
        ENDIF
      END FUNCTION W
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      FUNCTION dW(i,j)
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This function evaluates the differential of kernel
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        INCLUDE 'param.inc'
        REAL sig, rij, q, dW
        rij=0.0
        DO k=1, dim
          rij=rij+(pos(i,k)-pos(j,k))**2
        END DO
        rij=SQRT(rij)
        htest=(hh(i)+hh(j))/2.0
        IF(hmin /= hh(i) .AND. hmin /= htest)
     &                 CALL TERROR('KERNEL: hmin is inconsistent')
        q=rij/hmin
        ! Gauss kernel
        IF(skf == 1)THEN
          sig=(1.0/3.14)**(dim/2.0)
          IF(q <= 3.0)THEN
            dW=-(2.0*q*EXP(-q*q))/hmin
          ELSE
            dW=0.0
          ENDIF
          dW=sig*dW/hmin**dim
        ! spline-base kernel
        ELSEIF(skf == 2)THEN
          IF(dim ==1) sig=2.0/3.0
          IF(dim ==2) sig=10.0/(7.0*3.14)
          IF(dim ==3) sig=1.0/3.14
          IF(q <= 1.0)THEN
            dW=(-3.0*q+2.25*q**2)/hmin
          ELSEIF(q > 1.0 .AND. q <= 2.0)THEN
            dW=-0.75*(2.0-q)**2/hmin
          ELSEIF(q > 2.0)THEN
            dW=0.0
          ENDIF
          dW=sig*dW/hmin**dim
        ! Quintic kernel
        ELSEIF(skf == 3)THEN
          IF(dim ==1) sig=1.0/120.0
          IF(dim ==2) sig=7.0/(480.0*3.14)
          IF(dim ==3) sig=1.0/(120.0*3.14)
          IF(q <= 1.0)THEN
            dW=(-5.0*(3.0-q)**4+30.0*(2.0-q)**4-75.0*(1.0-q)**4)/hmin
          ELSEIF(q > 1.0 .AND. q <= 2.0)THEN
            dW=(-5.0*(3.0-q)**4+30.0*(2.0-q)**4)/hmin
          ELSEIF(q > 2.0 .AND. q <= 3.0)THEN
            W=(-5.0*(3.0-q)**4)/hmin
          ELSEIF(q > 3.0)THEN
            dW=0.0
          ENDIF
          dW=sig*dW/hmin**dim
        ENDIF
      END FUNCTION dW
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
      SUBROUTINE TERROR(message)
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!     This subroutine stops the program if there is any error
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
        CHARACTER(*) message
        WRITE(*,*) '*********!?!?!?! error:!?!?!?!*********'
        WRITE(*,*) message
        WRITE(*,*) '*********program is terminated*********'
        STOP
      END SUBROUTINE TERROR
!=====================================================================
!/////////////////////////////////////////////////////////////////////
!=====================================================================
!=====================================================================
!*************************   param.inc   ***************************
! ===================================================================
!     this file contains common definitions and parameters
! ===================================================================
      ! dimension and maximum number of particles
      INTEGER dim
             ! dim--> number of spatial dimensions (1, 2, or 3)
      PARAMETER(dim=1)
      INTEGER maxn
             ! maxn--> maximum number of SPH particles
      PARAMETER(maxn=5000)
      INTEGER nbody
             ! nbody--> number of SPH particles
      REAL ul, ut, um
             ! ul--> unit of length (m)
             ! ut--> unit of time (s)
             ! um--> unit of mass (kg)
             ! ud--> unit of density (kg/m^3)
             !   =um/ul^3
             ! uv--> unit of velocity (m/s)
             !   =ul/ut
             ! ub--> unit of magnetic field (tesla)
             !   =SQRT(um/ul)/ut
             ! G0--> gravitational constant
             !   =(6.68e-11*um/ul)*(ut/ul)**2
      COMMON /main/ nbody, ul, ut, um
      ! switches for different scenarios
      INTEGER isorad
             ! isorad--> isothermal or adiabatic shock?
             !       =1: adiabatic
             !       =2: isothermal
      INTEGER skf
             ! skf--> smoothing kernel function
             !    =1: Gauss kernel (Gingold & Monaghan 1981)
             !    =2: spline-base kernel (Monaghan 1985)
             !    =3: Quintic kernel (Morris 1997)
      INTEGER nnssl
             ! nnssl--> nearest neighbor search and smoothing length
             !      =1: tree with h=hfac*(mass/den)**(1/dim)
             !      =2: tree with dh/dt=(h/dim)*divvel
             !      =3: tree with fixed neighbors between max and min
      INTEGER dsm
             ! dsm--> density summation method
             !    =1: summation model without continuity
             !    =2: use continuity equation
      COMMON /senar1/ isorad, skf, nnssl, dsm
      ! tolerance and correction parameters
      REAL alphas, betas
             ! alphas--> shear viscosity
             ! betas--> bulk viscosity
      REAL epsi, eta
             ! epsi--> parameter in XSPH correction of velocities
             ! eta--> parameter to avoid singularities in viscosity
      PARAMETER(epsi=0.5, eta=0.1)
      REAL theta, eps
             ! theta--> tolerance parameter in tree structure
             ! eps--> tolerance parameter in tree structure
      PARAMETER(theta=0.25, eps=1.0e-4)
      REAL cour
             ! cour--> Courant number in step-time
      PARAMETER(cour=0.25)
      COMMON /toleran/ alphas, betas
      ! tree structure data arrays
      INTEGER nsubc, nquad
             ! nsubc--> number of descendants per cell
             ! nquad--> number of independent quadrupole components
      PARAMETER(nsubc=2**dim, nquad=2*dim-1)
      INTEGER inode
             ! inode--> initial number of nodes (bodies + cells)
      PARAMETER(inode=maxn+nsubc*maxn)
      INTEGER mxcell, node, incell, ncell
             ! mxcell--> number of cells in the system (=nsubc*nbody)
             ! node--> number of nodes (bodies + cells)
             ! incell--> index of first cell in arrays (=nbody+1)
             ! ncell--> number of cells currently in use (<=mxcell)
      INTEGER subp(inode,nsubc), root, sortind(maxn*nsubc)
             ! subp--> descendent of each cell
             ! root--> index of cell representing root (=incell)
             ! sortind(maxn*nsubc)--> sorted cells in descending size
      REAL mid(inode,dim), clsize(inode)
             ! mid--> geometric center of each cell
             ! clsize--> size of each cell
      REAL rcrit2(inode), quad(inode,nquad)
             ! rcrit2(incell:node)--> critical distances^2 of each cell
             ! quad(incell:node,nquad)--> quad moments of each cell
      COMMON /tree1/ mxcell, node, incell, ncell, sortind
      COMMON /tree2/ rcrit2, quad, subp, root, mid, clsize, ndesc
      ! neighbor search parameters and smoothing length
      INTEGER neighb(maxn), neighblist(maxn,maxn)
             ! neighb(maxn)-> number of neighbors for each particle
             ! neighblist(maxn,maxn)--> list of neighbors
      REAL hh(maxn), hhdot(maxn)
             ! hh(maxn)--> smoothing lengths of SPH particles
             ! hhdot(maxn)--> smoothing length rate
      REAL hmin
             ! hmin--> mean smoothing length of two neighbor particle
      COMMON /neighbor1/ neighb, neighblist, hh, hhdot, hmin
      ! states of SPH particles
      REAL pos(inode,dim), mass(inode), den(maxn)
             ! pos(node,dim)--> positions of bodies and cells

             ! mass(node)--> mass of bodies and cell
             ! den(maxn)--> density at position of each particle
      REAL vel(maxn,dim), divvel(maxn)
             ! vel(maxn,dim)--> velocities of each body
             ! divvel(maxn)--> divergence of velocity

      REAL acc(maxn,dim), dendot(maxn)
             ! acc(maxn,dim)--> acceleration of bodies
             ! dendot(maxn)--> density rate
      REAL sound(maxn), pre(maxn), temp(maxn)
             ! sound(maxn)--> sound speed of particles
             ! pre(maxn)--> pressure of particles
             ! temp(maxn)--> temperature of particles
      REAL u(maxn), udot(maxn)
             ! u(maxn)--> energy of particles
             ! udot(maxn)--> energy rate
      COMMON /state1/ pos, mass, vel, divvel, acc, dendot, u, udot
      COMMON /state2/ den, sound, pre, temp
      ! time integration parameters
      REAL tnow, dtmin
             ! tnow--> current time
             ! dtmin--> minimum time-step
      COMMON /time/ tnow, dtmin
!=====================================================================
!=====================================================================
```